

1. PowerPC SIMD Vectorization

- Synonyms:
 1. *AltiVec* - Motorola's trademark name
 2. *Velocity Engine* - Apple's marketing term
 3. *VMX* - IBM's original internal code name
 - SIMD, Vector Processor
- Has no unaligned load/store
 - Even elt load goes into varying location in vector!
- Has C interface as well as assembly
 - Apple intf diff from gcc, but apple calls their compiler gcc too!

2. OS X AltiVec Register Usage

REGISTER	USAGE	CALLEE SAVE
v0-v1	General use	NO
v2-v13	Parameter passing	NO
v14-v19	General	NO
v20-v31	General	YES
VRSAVE	vregs OS must maintain	YES
VSCR	Vector Status & Control	??

3. OS X Calling Sequence and Stack Frame

- Stack grows downward
- Caller puts callees' args in its frame
- Frame 16-byte aligned
- Caller saves LR and CR only if it wants to
- 8 word mandatory para area not init (in reg); there for callee use
- Has 224-byte red zone

Stack frame passed to callee

	fp reg save area (optional)
	ireg save area (optional)
	VRSAVE save word
	padding (optional)
	Vector register save
	Local storage (optional)
24(r1)	Parameter area (≥ 8 words)
20(r1)	TOC save area
16(r1)	Link editor doubleword
12(r1)	Compiler doubleword
8(r1)	Link register (LR) save
4(r1)	Condition register (CR) save
0(r1)	ptr to callee's stack

4. Special Vector Registers

VRSAVE, indicates what vector registers must be saved during context switch:

- Read by : `mfspr rx, VRsave`
- Written : `mtspr VRsave, rx`
- If Big endian bit i is 1, v_i must be saved, else not
- Callee's `vrsave` init to caller's & `0xFFF` (keep only callee-saved regs, `v20 – 31`), then add your usage

VSCR, controls IEEE compliance & saturation:

- Read by : `mfsvr vx`
- Written : `mtsvr vx`
- NJ at bit 15 big-endian, bit 16 little endian in VSCR
- at little endian 16 in `vx` (111 big end)
- Set to 0 to get ieee compliance!

5. PowerPC Vector Addressing

- Only indexed allowed : rx , ry
 - $@ = rx + ry$
 - $r0$ interp as 0 (except when ry):
- All @ truncated to 16-byte alignment
 - $EA = ((rx+ry) / 16)*16$
- Suffix indicates what elt size is:
 - '**b**' : byte
 - '**h**' : 2 bytes
 - '**w**' : 4 bytes
- Even elt-wise ld/st set/store only elt at VR EA % 16 (big endian)
 - For elt store, easiest to splat word thru entire vx
 - For elt ld, probably use `lvs1` and permute to get in low word

6. PPC AltiVec Transfer Operations

Mnemonic	Operands	Action
lvx	vz, rx, ry	$vz_{0-127} = *(rx+ry)$
lvxl	vz, rx, ry	$vz_{0-127} = *(rx+ry)$, mark LRU
lve[b,h,w]x	vz, rx, ry	Load elt $*(rx+ry)$ into @%16 elt vz , other elts left undefined!
stvx	vz, rx, ry	$*(rx+ry) = vz$
stve[b,h,w]x	vz, rx, ry	Store elt @%16 of vz to $*(rx+ry)$
vmr	vz, vx	$vz = vx$ (<i>register copy</i>)
vspltis[b,h,w]	$vz, imm5$	set all elts of vz to val [-16,15]
vspltiu[b,h,w]	$vz, imm5$	set all elts of vz to val ($\text{sign}(imm5) \ll \text{sizeof}(elt) + imm5$ (e.g. $256 + (-15) = 240$ for $imm5 = -15$)

- Setting rx as $r0$ substitutes zero for rx .

7. Common AltiVec Bit-Level Operations

Mnemonic	Operands	Action
vand	vz, vx, vy	$vz = vx \& vz$
vandc	vz, vx, vy	$vz = vx \& (\sim vz)$
vor	vz, vx, vy	$vz = vx vz$
vxor	vz, vx, vy	$vz = vx \hat{=} vz$
vnor	vz, vx, vy	$vz = \sim(vx vz)$
vsel	vz, vv, vx, vy	$\forall \text{ bit } i : vz_i = (vy_i ? vx_i : vv_i)$
vrl[b,h,w]	vz, vx, vy	each elt of vz rotated left by bits spec corresp. in elt of vy
vsl[b,h,w]	vz, vx, vy	each elt of vz shifted left by bits spec corresp. in elt of vy

- Have `vsr[b,h,w]` for logical shifts
- Have `vsar[b,h,w]` for arithmetic (sign bit maintaining) shifts

8. Common AltiVec Floating Point Computation Instructions

Mnemonic	Operands	Action
<code>vmaddfp</code>	vz, vv, vx, vy	$vz = vv * vx + vy$
<code>vnmsubfp</code>	vz, vv, vx, vy	$vz = -(vv * vx - vy)$
<code>vaddfp</code>	vz, vx, vy	$vz = vx + vy$
<code>vsubfp</code>	vz, vx, vy	$vz = vx - vy$
<code>vmaxfp</code>	vz, vx, vy	$vz = \max(vx, vy)$
<code>vminfp</code>	vz, vx, vy	$vz = \min(vx, vy)$
<code>vrefp</code>	vz, vx	$vz \approx 1.0/vx$

9. Common Regular Altivec Permute Operations

Mnemonic	Operands	Action
<code>vsldoi</code>	$vz, vx, vy, imm4$	concat vx and vy , shift word $imm4$ (0-15 bytes to left, store next 16 bytes in vz
<code>vmergh[b,h,w]</code>	vz, vx, vy	put high elt of vx in high elt of vz , get next word from high elt of vy , cont.
<code>vmergl[b,h,w]</code>	vz, vx, vy	put low elt of vy in low elt of vz , get next word from low elt of vx , cont.
<code>vsplt[b,h,w]</code>	$vz, vx, imm5$	fill all elts of vz with elt $imm5$ of vx

10. Common Irregular Altivec Permute Instructions

- Whole lot of these, usually use `vperm` with `lvs1`
- `vperm, vz, vv, vx, vy`
 - Each byte in `vz` set to byte number (big endian) stored in corresp. byte of `vy`, where `vv` has bytes 0x00-0x0E, and low order byte of `vx` has bytes 0x10-0x1F.
- `lvs1 vz, rx, ry`
 - Create `vperm` map in `vz` that will align $EA=rx+ry$
 - Get identity map by using EA that is aligned (eg., `r0, r1`)

```
; N unalign lds, N-1 aligned ops
lvs1   v9, r3, r2
lvx    v0, r3, r2
addi   r2, r2, 16
lvx    v1, r3, r2
addi   r2, r2, 16
lvx    v2, r3, r2
addi   r2, r2, 16
lvx    v3, r3, r2
vperm  v4, v1, v2, v9
vperm  v5, v2, v3, v9
vperm  v6, v3, v4, v9
```

```
; Load & dup scalar
lvs1   v9, r1, r2
lvewx  v0, r1, r2
vspltw v9, v9, 0
vperm  v0, v0, v0, v9
```

11. SASUM in Altivec Assembly (1 of 2)

```
#define Mjoin(pre, nam) my_join(pre, nam)
#define my_join(pre, nam) pre ## nam
#define N          r3
#define X          r4
#define rtmp       r5
#define lrsav      r6
#define vrsav      r7
#define rtmp2      r8
#define II         r9
#define rsum       f1
#define rX         f2
#define vabs       v0
#define vX         v1
#define vsum       v2
/*          r3          r4          r5
float ATL_UASUM(int N, float *X, const int incX) */
    .text
    .globl Mjoin(,ATL_UASUM)
Mjoin(,ATL_UASUM):
    mfspr    vrsav, VRsave
    andi.   rtmp, vrsav, 0xFFF
    li      rtmp2, 3          ; my vreg usage
    sldi    rtmp2, rtmp2, 30
    or      rtmp, rtmp, rtmp2
    mtspr   VRsave, rtmp
    mflr    lrsav

    xor     rtmp, rtmp, rtmp
    stw     rtmp, -4(r1)
    lfs     rsum, -4(r1)    ; rsum = 0
    lis     rtmp, ha16(DONE)
    addi    rtmp, rtmp, lo16(DONE)
    mtlr    rtmp
    cmpwi   0, N, 8
    blt-    CLEANUP
    vspltisw vabs, 1
    vspltisw vX, 15
    vadduwm vX, vX, vX      ; vX = 30
    vadduwm vX, vabs, vX    ; vX = 31
    vslw    vabs, vabs, vX
    vnor    vabs, vabs, vabs; vabs =
    vxor    vsum, vsum, vsum; vsum =
    addi    rtmp, X, 15
    srldi   rtmp, rtmp, 4
    sldi    rtmp, rtmp, 4
    subf.   rtmp, X, rtmp
    bnel-   FORCEALIGN
```

12. SASUM in Altivec Assembly (2 of 2)

```
srldi    II, N, 2
sldi     II, II, 2 ; I=(N/4)*4
subf     N, II, N
sldi     II, II, 2
add      X, X, II
neg      II, II
LOOP4:
lvx      vX, X, II
vand     vX, vX, vabs
vaddfp   vsum, vsum, vX
addic.   II, II, 16
bne+     LOOP4
cmpwi    0, N, 0
bnel-    CLEANUP
DONE:
; vsum = {sum0, sum1, sum2, sum3}
vsldoi   vX, vsum, vsum, 8
; vX    = {sum2, sum3, sum0, sum1}
vaddfp   vsum, vsum, vX
; vsum = {sum02, sum13, sum02, sum13}
vsldoi   vX, vsum, vsum, 4
; vX    = {sum13, sum02, sum02, sum13}
vaddfp   vsum, vsum, vX
; vsum = {sum0123, sum0123, sum02, sum13}
li       II, -16
stvevx   vsum, r1, II
lfsx     rX, r1, II
fadds    rsum, rsum, rX
mtlr     lrsav
mtspr    VRsave, vrsav
blr
FORCEALIGN:
srldi    rtmp2, rtmp, 2
subf     N, rtmp2, N
add      X, X, rtmp
neg      rtmp, rtmp
bl       LOOP
CLEANUP:
sldi     rtmp, N, 2
add      X, X, rtmp
neg      rtmp, rtmp
LOOP:
lfsx     rX, X, rtmp
fabs     rX, rX
fadds    rsum, rsum, rX
addic.   rtmp, rtmp, 4
bne+     LOOP
blr
```

13. Unaligned AltiVec SAXPY (1 of 2)

```

;          r3          f1:r4          r5          r6          r7          r8
;void ATL_UAXPY(int N, float alpha, float *X, int incX, float *Y, int incY)
#define N          r3          li          II, -16
#define X          r5          stfsx     ralpha, r1, II
#define Y          r7          lvewx     valpha, r1, II
#define II         r4          vspltw    valpha, valpha, 0
#define Xn         r6          ; Use cleanup unless 3 vec iter
#define vrsav      r8          lis        II, ha16(DONE)
#define lrsav      r0          addi       II, II, lo16(DONE)
#define ralpha     f1          mtlr      II
#define rX         f2          mr         II, N
#define rY         f3          cmpwi     0, N, 12
#define vX         v0          blt-     CLEANUP
#define vY         v1          ; Force Y alignment
#define vXn        v2          addi       II, Y, 15
#define valpha     v3          srwi      II, II, 4
#define vmap       v4          slwi      II, II, 4
          .text              sub.      II, II, Y      ; II = Y16 - Y
          .globl Mjoin(,ATL_UAXPY)
          bnel-  FORCEALIGN
Mjoin(,ATL_UAXPY):          ; Compute # vec iters
          mfspr   vrsav, VRsave          srwi      II, N, 2
          mflr   lrsav          addi      II, II, -1      ; II = (N/4*4)-1
; Signal my vreg usage          slwi      II, II, 2      ;
          andi.   II, vrsav, 0xFFF          sub       N, N, II      ; Nr = N-N4-4
          oris   II, II, 0xF800
          mtspr   VRsave, II

```

14. Unaligned AltiVec SAXPY (2 of 2)

```

slwi    II, II, 2    ; II = N4*sizeof()    ; On entry, II has # of bytes to aligned Y
add     X, X, II    FORCEALIGN:
add     Y, Y, II    srwi    II, II, 2    ; II /= sizeof()
addi    Xn, X, 16   ; Next elt of X    ; On entry, II has # of iters to clean
neg     II, II    CLEANUP:
lvsl    vmap, X, II ; perm pattern for X  slwi    II, II, 2
lvx     vX, X, II  add     X, X, II
LOOP4U:  add     Y, Y, II
lvx     vXn, Xn, II neg     II, II
vperm   vX, vX, vXn, vmap  LOOP1:
lvx     vY, Y, II  lfsx    rX, X, II
vmaddfp vY, valpha, vX, vY  lfsx    rY, Y, II
vmr     vX, vXn  fmadds  rY, ralpha, rX, rY
stvx    vY, Y, II  stfsx   rY, Y, II
addic.  II, II, 16  addic.  II, II, 4
bne+    LOOP4U  bne+    LOOP1
mr      II, N  blr
bl      CLEANUP
DONE:
mtrl    lrsav
mtspr   VRsave, vrsav
blr

```